Human-Machine Collaborative Optimization via Apprenticeship Scheduling

Matthew Gombolay[*]
Massachusetts Institute of Technology
77 Massachusetts Avenue
Cambridge, Massachusetts 02139
gombolay@csail.mit.edu

Reed Jensen & Sung-Hyun Son
MIT Lincoln Laboratory
244 Wood Street
Lexington, MA 02420
{rjensen,sson}@ll.mit.edu

Julie Shah
Massachusetts Institute of Technology
77 Massachusetts Avenue
Cambridge, Massachusetts 02139
julie_a_shah@csail.mit.edu

Abstract

Scheduling techniques are typically developed for specific industries and applications through extensive interviews with domain experts to codify effective heuristics and solution strategies. As an alternative, we present a technique called Collaborative Optimization via Apprenticeship Scheduling (COVAS), which performs machine learning using human expert demonstration, in conjunction with optimization, to automatically and efficiently produce optimal solutions to challenging real-world scheduling problems. COVAS first learns a policy from human scheduling demonstration via apprenticeship learning, then uses this initial solution to provide a tight bound on the value of the optimal solution, thereby substantially improving the efficiency of a branch-and-bound search for an optimal schedule. We demonstrate this technique on a variant of the weapon-to-target assignment problem, and show that it generates substantially superior solutions to those produced by human domain experts, at a rate up to $\sim$ 10 times faster than an optimization approach that does not incorporate human expert demonstration.

Introduction

Scheduling is a costly problem for many industries, both with regard to the effort required to develop a solution technique and the time necessary to produce a schedule. However, attempts to take "shortcuts within the scheduling process may yield low-quality schedules that result in wasted resources. Traditionally, scheduling techniques are developed for specific industries and applications by consultants who conduct extensive interviews with the domain experts who manually or semi-manually perform scheduling tasks. The goal of these interviews is to codify effective heuristics and strategies for the problem, in order to then craft efficient automated scheduling techniques. This process is time-consuming and largely manual, because while domain experts can readily explain the key aspects or features of their problem-solving strategy, they are not typically able to precisely describe *how* they use those features when making decisions (Cheng, Wei, and Tseng 2006; Raghavan, Madani, and Jones 2006).

In this work, we propose Collaborative Optimization via Apprenticeship Scheduling (COVAS), an approach that incorporates machine learning from human expert demonstration, in conjunction with optimization, to automatically and efficiently produce optimal solutions to challenging real-world scheduling problems. Our method performs policy learning using a training dataset comprised of schedules demonstrated by humans, as well as a recently developed technique for apprenticeship scheduling (Gombolay et al. 2016a). In prior work, the technique was proposed in order to simply emulate human expert scheduling decisions; in this work, we use the apprenticeship scheduler to generate a favorable (if suboptimal) initial solution to a new scheduling problem. To guarantee that the generated schedule is serviceable, we augment the apprenticeship scheduler to solve a constraint satisfaction problem, ensuring that the execution of each scheduling commitment does not directly result in infeasibility for the new problem. COVAS uses this initial solution to provide a tight bound on the value of the optimal solution, substantially improving the efficiency of a branch-and-bound search for an optimal schedule.

We demonstrate our approach by solving a real-world anti-ship missile defense problem, and report that COVAS produces substantially superior solutions to those produced by human domain experts, at a rate twice as fast as an optimization approach that does not incorporate human expert demonstration.

Related Work

Recent research has aimed to capture goal-based knowledge obtained through demonstration via a process known as reward learning (Abbeel and Ng 2004; Berry et al. 2011; Ijspeert, Nakanishi, and Schaal 2002; Konidaris, Osentoski, and Thomas 2011; Zheng, Liu, and Ni 2015; Odom and Natarajan 2015; Terrell and Mutlu 2012; Thomaz and Breazeal 2006; Vogel et al. 2012; Ziebart et al. 2008). Inverse reinforcement learning (IRL), the most common approach, learns a reward function to capture the intent of the demonstrators and then trains a policy via reinforcement learning to maximize that reward function. However, as

[*]This material is based upon work supported under Air Force Contract No. FA8721-05-C-0002 and/or FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the U.S. Air Force.

noted in prior works (Gombolay et al. 2016a; Wu et al. 2011; Wang and Usher 2005; Zhang and Dietterich 1995), the large amount of data required to regress over the large state spaces associated with scheduling problems remains daunting, and RL-based scheduling solutions exist only for simple problems (Wu et al. 2011; Wang and Usher 2005; Zhang and Dietterich 1995).

An alternate approach specially designed for meeting scheduling (Berry et al. 2011) requires users to complete an extensive questionnaire in order to solicit their preferences for scheduling meetings. This technique then maps those preferences to an objective function and solves for the optimal meeting schedule via a mixed-integer linear program (MILP). However, this approach is limited to small problems that could be efficiently solved as an integer linear program (Berry et al. 2011). State-of-the-art techniques for solving scheduling problems with complex temporal constraints via integer linear programs are limited to problems involving five agents and 50 tasks, at most (Ciré, Coban, and Hooker 2013).

Another approach, called policy learning, focuses on learning a mapping from states to actions (Chernova and Veloso 2007; Huang and Mutlu 2014; Sammut et al. 1992; Ramanujam and Balakrishnan 2011). This technique has been applied to learn cognitive decision-making tasks from human experts, such as determining an airport runway configuration (Ramanujam and Balakrishnan 2011). Similarly, the learning system AlphaGo incorporates an initial policy-leaning phase (Silver et al. 2016). The AlphaGo framework began by solving a supervised policy learning problem to imitate the decision-making of human Go players. AlphaGo's policy was then improved through self-play using a policy gradient algorithm (Sutton et al. 1999). This approach is promising for solving scheduling problems by learning policies through expert demonstration. However, we are unaware of any prior attempts to apply policy learning to the scheduling domain, other than work by Gombolay et al. (Gombolay et al. 2016a), which aimed to simply emulate human expert scheduling policies, rather than improve upon them. Techniques that rely upon function approximation and policy gradient descent or variants of q-learning (such as the framework employed by AlphaGo) are less desirable for scheduling applications, as it is inefficient to specify or learn complex temporal constraints which are often non-Markovian and the solution techniques are only guaranteed to converge to a local optimal solution.

A small number of prior works have pursued approaches outside of the family of techniques for policy learning. Some directly modeled the trustworthiness of the demonstrations via robust Bayesian inverse reinforcement learning (Zheng, Liu, and Ni 2015). For example, Zheng et al. showed that their approach was better able to capture the ground-truth objective function from imperfect training data than regular, Bayesian IRL (Ramachandran and Amir 2007), which does not include a trustworthiness parameter for demonstrations. Zheng et al. validated their approach using a synthetic dataset in an experiment with the goal of identifying the best route through an urban domain.

Banerjee et al. addressed a domain in which the system was required to repeatedly solve a scheduling problem wherein the variables remained the same, but the constraints for those variables changed (Banerjee et al. 2011). Using a MILP formulation, they proposed a machine learning-optimization pipeline in which the system performed a branch-and-bound search over the integer variables, and used the prediction of a regression algorithm trained on examples of previously solved problems to provide a provable lowerbound on the optimality of the current integer variable assignments. A shortcoming of this approach is its reliance upon the ability to generate a large database of solutions to train the regression algorithm. This generation requires the costly exercise of repeatedly solving a large set of MILPs.

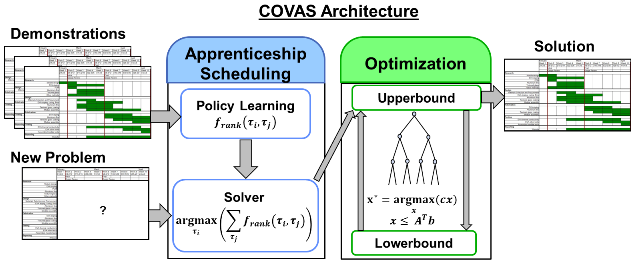

Figure 1: The COVAS architecture.

The technique presented in this paper was inspired by these prior works, which synthesize machine learning techniques, optimization and human demonstration. To our knowledge, our work is the first to develop and demonstrate an approach to learning through human demonstrations to efficiently produce optimal solutions for complex real-world scheduling problems. Our method employs a policy learning phase to learn from human demonstration, and uses the resulting policy as an initial solution to provide a tight bound on the value of the optimal solution. We show that this policy can be used in conjunction with a MILP solver to substantially improve the efficiency of a branch-and-bound search for an optimal schedule. Our work is distinguished from prior works that incorporated policy gradient descent or variants of q-learning in that COVAS is guaranteed to produce a globally optimal solution to the scheduling problem. Also, COVAS can be employed as an anytime algorithm that provides a bound on the sub-optimality of the solution.

Model for Collaborative Optimization via Apprenticeship Scheduling

Here, we provide an overview of the COVAS architecture, and then present its two components: the policy learning and optimization routines.

COVAS Architecture

Figure 1 depicts an overview of the COVAS framework.

The system takes as input a set of domain expert scheduling demonstrations (e.g., Gantt charts, as shown in Figure 1) that contains information describing which agents complete which tasks, when and where. These demonstrations are passed to an apprenticeship scheduling algorithm that learns a classifier, $f_{priority}(\tau_i, \tau_j)$, to predict whether the demonstrator(s) would have chosen scheduling action τ_i over action $\tau_j \in \boldsymbol{\tau}$.

Next, COVAS uses $f_{priority}(\tau_i, \tau_j)$ to construct a schedule for a new problem. COVAS creates an event-based simulation of this new problem and runs the simulation in time until all tasks have been completed. In order to complete tasks, COVAS uses $f_{priority}(\tau_i, \tau_j)$ at each moment in time to select the best scheduling action to take. We describe this process in detail in the next section.

Next, COVAS provides this output as an initial seed solution to an optimization subroutine (i.e., a MILP solver).The initial solution produced by the apprenticeship scheduler improves the efficiency of a search by providing a bound on the objective function value of the optimal schedule.

Here, we briefly review the basic technique for solving a MILP for a full overview, we refer the reader to (Bertsimas and Weismantel 2005). In general, solving a MILP requires iteratively identifying ever-tighter upper- and lowerbounds for a given problem in order to inform a branch-and-bound search over the integer variables. To find an upperbound, one must satisfy the constraints of the MILP: $Ax \leq b$. To identify a lowerbound, one can solve a linear relaxation of the problem.Such a relaxation can be computed quickly; however, it rarely results in a feasible solution. As each new upper- and lowerbound solution is found, the algorithm is able to prune areas of the search tree and focus its search on areas that can yield the optimal solution. After the algorithm has identified an upper- and lowerbound within some threshold, COVAS returns the solutions that have been proven optimal within that threshold. Thus, an operator can use COVAS as an anytime algorithm and terminate the optimization upon finding a solution that is acceptable within a provable bound.

Apprenticeship Scheduling Subroutine

In this section, we review the apprenticeship scheduling subroutine for COVAS. Our approach incorporates policy learning using a training dataset comprised of schedules demonstrated by humans, as well as a recently developed technique for apprenticeship scheduling. The apprenticeship scheduling algorithm (Gombolay et al. 2016a) takes as input demonstrations in which human experts manually solve randomly generated variants of a real-world scheduling problem. The apprenticeship scheduler has been shown in empirical evaluation to learn a policy that effectively emulates the human expert scheduling policy in a new problem variant. In this work, we use the apprenticeship scheduler to generate a favorable (if suboptimal) initial solution to a new scheduling problem. To guarantee that the generated schedule is serviceable, we augment the apprenticeship scheduler to solve a constraint satisfaction problem in order to ensure that each scheduling commitment does not directly result in infeasibility for the new problem through the execution of that action.

Consider a scheduling problem containing a set of tasks $\mu \in M$, agents $a \in A$ and locations to complete tasks $x \in X$, as well as a set of scheduling actions taken at each moment in time $\tau_i = \langle \mu, a, x, t \rangle$. For each action taken, τ_i, the learning system can also compute the set of actions not taken, $\tau_j \in \boldsymbol{\tau}$. Each tuple has an associated real-valued feature vector, γ_{τ_i}. Features of this vector may include the deadline for μ, the distance from the agent's current location to x or how quickly the agent is able to complete the task. The system allows, for a given moment in time, all possible tuples to share a common pointwise feature vector, ξ_t, which captures features that are not well described by pairwise comparisons, such as the proportion of completed tasks or of idle agents.

$$
{}^{priority}\theta^{m}_{\langle \tau_i, \tau_x \rangle} := \left[\xi_{\boldsymbol{\tau}}, \gamma_{\tau_i} - \gamma_{\tau_x} \right], y^{m}_{\langle \tau_i, \tau_x \rangle} = 1,
$$
$$
\forall \tau_x \in \boldsymbol{\tau} \backslash \tau_i, \forall O_m \in \boldsymbol{O} | \tau_i \text{ scheduled in } O_m \tag{1}
$$
$$
{}^{priority}\theta^{m}_{\langle \tau_x, \tau_i \rangle} := \left[\xi_{\boldsymbol{\tau}}, \gamma_{\tau_x} - \gamma_{\tau_i} \right], y^{m}_{\langle \tau_x, \tau_i \rangle} = 0,
$$
$$
\forall \tau_x \in \boldsymbol{\tau} \backslash \tau_i, \forall O_m \in \boldsymbol{O} | \tau_i \text{ scheduled in } O_m \tag{2}
$$

These vectors serve to create the training data, as shown in Equations 1-2. For each scheduling observation (i.e., a specific time point within a schedule), the system creates a set of positive and negative examples. For each such moment, we take the feature vector of the action taken, γ_{τ_i}; less the feature vector of an action not taken, γ_{τ_j}; concatenate to that difference the pointwise vector, $\xi_{\boldsymbol{\tau}}$; and, given that example, a positive label (Equation 1). To create a negative example, we take the feature vector of an action not taken, γ_{τ_j}; less the feature vector of the action taken, γ_{τ_i}; and concatenate to that difference the pointwise vector, $\xi_{\boldsymbol{\tau}}$ (Equation 2). We create one positive and negative example for each action not taken, τ_j, for each observation. Finally, we train a classifier on these examples to learn a priority function, $f_{priority}(\tau_i, \tau_j)$, in order to predict whether scheduling action τ_i is better or worse than τ_j. The computational complexity of the algorithm vis-á-vis Equation 3 is $O(|\boldsymbol{\tau}|^2 d)$ per time step, where d is the maximum depth of the decision tree (Gombolay et al. 2016a).

In this work, the learned policy $f_{priority}(\tau_i, \tau_j)$ is applied to obtain the initial solution to a new scheduling problem as follows: First, the user must instantiate a simulation of the scheduling domain; then, at each time step in the simulation, take the scheduling action predicted by Equation 3 to be the action that the human demonstrators would take. This equation identifies the task τ_i with the highest importance marginalized over all other tasks $\tau_j \in \boldsymbol{\tau}$.

Each selected action is then validated using a schedulability test (i.e., solving a constraint satisfaction problem) to ensure that direct application of that action does not violate the constraints of the new problem. For example, in antiship missile defense, one would check to ensure that the action does not result in a suicidal deployment (i.e., the decoy directly causes a missile to impact the ship). The test must be designed to be fast (e.g., polynomial complexity) so as to make the benefit to feasibility and optimality in the resulting schedule worth the additional complexity. If, at a given time step, τ_i^* does not satisfy the schedulability test, COVAS uses Equation 3 for all $\tau_i \in \boldsymbol{\tau} \backslash \tau_i^*$ in order to consider the second-best action. If no action $\tau_i \in \boldsymbol{\tau}$ passes the schedulability test, no action is taken during that time step.

While the schedulability test forces the apprenticeship scheduling algorithm to follow a subset of the full constraints in the MILP formulation, it is possible that the algorithm my not successfully complete all tasks. However, our MILP formulation is flexible in such cases, as we present in the next section. Here, we model tasks as optional and use the objective function to maximize the total number of tasks completed. In turn, constraints for a task that the apprenticeship scheduling algorithm did not satisfactorily complete can be turned off, with a corresponding penalty in the objective function score. Thus, an initial seed solution that has not completed all tasks (i.e., satisfied all constraints to complete the task) can still be helpful for seeding the MILP.

$$\tau_i^* = \underset{\tau_i \in \boldsymbol{\tau}}{\operatorname{argmax}} \sum_{\tau_x \in \boldsymbol{\tau}} f_{priority}(\tau_i, \tau_x) \tag{3}$$

Optimization Subroutine

For optimization, we employ mathematical programming techniques to solve mixed-integer linear programs via branch-and-bound search. COVAS incorporates the solution produced by the apprenticeship scheduler to seed a mathematical programming solver with an initial solution. This is a built-in capability provided by many off-the-shelf, state-of-the-art MILP solvers, including CPLEX[1] and Gurobi[2]. This seed provides a tight bound on the value of the optimal solution, which serves to dramatically cut the search space, allowing the system to more quickly hone in on the area containing the optimal solution and, in turn, more quickly solve the optimization problem. Furthermore, this approach allows COVAS to quickly achieve a bound on the optimality of the solution provided by the apprenticeship scheduling subroutine. In such a manner, an operator can determine whether the apprenticeship scheduling solution is acceptable or whether waiting for successive solutions is warranted.

Methodology for Evaluation of COVAS with a Real-World Scheduling Domain

Here, we demonstrate COVAS in the context of a real-world anti-ship missile defense (ASMD) problem. First, we formally define the problem a variant of the well-studied weapon-to-target assignment problem (Ahuja et al. 2007) and outline its usefulness as an appropriate test domain for COVAS.

Overview of Anti-Ship Missile Defense Problem

In ASMD, the goal is to protect one's naval vessel against attacks by heterogeneous anti-ship missiles. Recent technological advances in electronic warfare have prompted the development of what are known as "soft-kill weapons (i.e., decoys) that mimic the qualities of a target in order to direct the missile away from its intended destination.

Developing tactics for soft-kill weapon coordination is highly difficult due to the relationship between missile behavior and the characteristics of soft-kill weapons. The control laws governing anti-ship missiles are varied, and the captain must select the correct decoy types in order to counteract the associated anti-ship missiles. Further, decoys have different financial costs and timing characteristics: Some decoys, such as unmanned aerial vehicles (UAVs), are able to function during the entire engagement, while others, such as an infrared (IR) flares, evaporate after a certain time. In turn, a captain may be required to use multiple decoys in tandem in order to divert a single anti-ship missile. Moreover, there is a complex interplay between the types and locations of decoys relative to the control laws governing anti-ship missiles. For example, deployment of a particular decoy, while effective against one airborne enemy missile, may actually cause a second enemy missile that was previously homing in on a second decoy to now impact the ship when it would have missed otherwise.

The ASMD problem is characterized as the most complex class of scheduling problem according to the Korsah et al. taxonomy (Korsah, Stentz, and Dias 2013): **XD [MA-MT-TA]**. The problem considers multi-task agents (MA) in the form of decoys, each of which can work to divert multiple missiles at the same time. The problem also considers multi-agent tasks (MT): a feasible solution may require the simultaneous use of multiple agents in order to complete an individual task. Further, time-extended agent allocation (TA) must be taken into consideration, given the potential future consequences of scheduling actions taken at the current moment. Finally, the ASMD problem falls within the XD class, because each task may be decomposed in a variety of ways each with their own cost in order to accomplish the same goal, and each decomposition affects the value and feasibility of the decompositions of other tasks.

ASMD Problem Formulation

The ASMD can be formally modeled as follows: We first define a task $\mu \in M$ as the job of defending a ship against an individual missile; successfully completing all tasks results in diverting all enemy missiles. We also define an agent $d \in D$ (i.e., a decoy) as an actor used to aid in accomplishing tasks. A scheduling assignment is then represented by a four-tuple $\langle d, \mu, \vec{x}, t \rangle$, where d is a decoy, m is the associated missile, $\vec{x}$ is the relative location (in Cartesian coordinates) of the decoy relative to the ship and t is the moment when the decoy should be deployed.

As ASMD is a time-extended problem, the formulation must discretize time. However, note that the granularity with which the task of protecting the ship from a given missile is decomposed as a function of time is a modeling choice with ramifications for the quality and computation time of a solution. Consider a missile that will hit a ship if it tracks a missile in some time interval $[t, t')$ for a duration $dt = t - t'$. The captain might, at time t, deploy a decoy d, such as a hovering UAV, that is able to last the entire duration dt. However, it may be preferable to deploy one or more decoys d', each of which remains active for a portion of the specified time interval. Furthermore, in a situation wherein another missile m' is launched before m, it may be best to have a decoy deployed before t that can divert both m and m' during part or all of those missiles' flights.

Because we do not know a priori the best time to deploy a decoy that can be used for varying portions (i.e., subtasks) of the task of mitigating each missile, we must decompose the

[1] IBM ILOG CPLEX Optimization Studio http://www-03.ibm.com/software/products/en/ibmilogcpleoptistud

[2] Gurobi Optimization, Inc. http://www.gurobi.com

task into sufficiently small time steps. Discretizing time exponentially increases the search space, and thus the time to compute the solution; therefore, there is a balance between optimality (and feasibility) and computation time. In order to generate an exact solution, we chose the least-common multiple of the time constants, which is trivially 1, as the unit of time in our simulation.

We formulate the ASMD as a mixed-integer linear program in Equations 4-25:

$$\min z, z = \alpha \sum_{d} c_d U_d - \alpha' \sum_{d,m,t} A_{d,m,t} - \alpha'' \sum_{m} V_m \tag{4}$$

$$A_{d,m,t} \leq A_{d,t}, \forall d, m, t \tag{5}$$

$$A_{d,m,t} \leq U_{d,m}, \forall d, m, t \tag{6}$$

$$X_{d,l} \leq U_d, \forall d, l \tag{7}$$

$$S_{d}^{decoy} \leq S_{d,m}^{decoy}, \forall d, m \tag{8}$$

$$S_{d,m}^{decoy} \leq t + M(1 - A_{d,m,t}), \forall d, m, t \tag{9}$$

$$F_{d,m}^{decoy} \leq F_{d}^{decoy}, \forall d, m \tag{10}$$

$$tA_{d,m,t} \leq F_{d,m}^{decoy}, \forall d, m, t \tag{11}$$

$$M(U_{d,m} - 1) \leq S_{d,m}^{decoy} - F_{d,m}^{decoy} - 1 + \sum_{t} A_{d,m,t} \leq M(1 - U_{d,m}) \tag{12}$$

$$M(U_d - 1) \leq F_{d}^{decoy} - S_{d}^{decoy} - dt_{d}^{evap} \leq M(1 - U_d) \tag{13}$$

$$\sum_{l} X_{d,l} \leq 1, \forall d \tag{14}$$

$$U_{d,m} \leq \sum_{l\,|\,m \text{ seduced by decoy } d \text{ in location } l} X_{d,l}, \forall d, m \tag{15}$$

$$1 = \sum_{d} A_{d,m,t} + \sum_{g} G_{g,m,t}, \forall m, t \tag{16}$$

$$\begin{gathered} t_{m}^{appear} - F_{d}^{decoy} \geq M(X_{d,l} + V_m - J_{d,m} - 2), \\ \forall d, l, m \text{ s.t. decoy d in location l would cause missile } m \text{ to impact the ship.} \end{gathered} \tag{17}$$

$$\begin{gathered} S_{d}^{decoy} - ETA_m \geq M(X_{d,l} + V_m + J_{d,m} - 3), \forall d, l, m \text{ s.t. decoy d} \\ \text{in location l would cause missile } m \text{ to impact the ship.} \end{gathered} \tag{18}$$

$$V_m \leq \sum_{d} A_{d,m,t}, \forall m, t | t \text{ in critical region for missile } m. \tag{19}$$

$$\begin{gathered} 2 \geq A_{d,m,t} + X_{d,l} + X_{d',l'}, \forall d, d', l, l', m, t \text{ s.t. missile m is more} \\ \text{attracted to decoy d' at location l' than decoy d at location l at time t.} \end{gathered} \tag{20}$$

$$\begin{gathered} 1 \geq A_{d,m,t} + A_{d',m,t}, \forall d, d', m, t \text{ s.t. } d \neq d' \\ \text{and t is in a critical region before impact.} \end{gathered} \tag{21}$$

$$S_{g,m}^{ship} \leq t + M(1 - G_{g,m,t}), \forall g, m, t \tag{22}$$

$$t * G_{g,m,t} \leq F_{g,m}^{ship}, \forall g, m, t \tag{23}$$

$$M(U_{g,m} - 1) \leq S_{g,m}^{ship} - F_{g,m}^{ship} - 1 + \sum_{t} G_{g,m,t} \leq M(1 - U_{g,m}) \tag{24}$$

$$\begin{aligned} F_{g,m}^{ship} - S_{g,m}^{ship} \geq{} & M(G_{g,m,t} - 1) \\ & + \begin{cases} dt_{m}^{re-target} - 1 & \text{if } t < ETA_m - dt_{m}^{re-target}, \\ ETA_m - t - 1 & \text{otherwise.} \end{cases} \\ & + \begin{cases} -MG_{g,m,t-1} & \text{if } t > t_{m}^{appear}, \\ 0 & \text{otherwise.} \end{cases} \\ & \forall g, m, t | t_{m}^{appear} \leq t < ETA_m \end{aligned} \tag{25}$$

This formulation incorporates a set of binary decision variables: $A_{d,m,t} \in \{0, 1\}$ is set to 1 to indicate that decoy d is assigned to missile m at time t, and is 0 otherwise. $A_{d,t} \in \{0, 1\}$ is set to 1 to indicate that decoy d is assigned to some missile at time t, and is 0 otherwise. $U_{d,m} \in \{0, 1\}$ is set to 1 to indicate that decoy d is used against missile m, and is 0 otherwise. $U_d \in \{0, 1\}$ is set to 1 to indicate that decoy d is used in the solution, and is 0 otherwise. $X_{d,l} \in \{0, 1\}$ is set to 1 to indicate that decoy d is deployed at location l, and is 0 otherwise. $V_m \in \{0, 1\}$ is set to 1 to indicate that missile m has been effectively diverted, and is 0 otherwise. $G_{g,m,t} \in \{0, 1\}$ is set to 1 to indicate that missile m is tracking the ship at time t. A single missile might have multiple, separate epochs during which it tracks the ship (e.g., it first tracks the ship, then tracks a decoy, then tracks the ship again after that decoy evaporates); thus, the program can choose which index g to represent the various epochs in $G_{g,m,t}$. $J_{d,m} \in \{0, 1\}$ is set to 1 to indicate that decoy d is deployed after missile m's flight (i.e., after it either hits the ship or is guided astray by a decoy).

The program contains the following set of continuous variables: $S_{d,m}^{decoy}$ represents the start time of the assignment of decoy d to missile m, and S_{d}^{decoy} is the time at which decoy d is deployed from the ship. Likewise, $F_{d,m}^{decoy}$ represents the finish time of the assignment of decoy d to missile m, and F_{d}^{decoy} is either the time at which the decoy evaporates or the end of the engagement. $S_{g,m}^{ship}$ indicates the start time of missile m tracking the ship during epoch g, and $F_{g,m}^{ship}$ indicates the finish time of missile m tracking the ship during epoch g.

The program also includes the following set of constants: $dt_{m}^{re-target}$ is the duration for which a missile will track a single target (i.e., decoy or ship) before re-assessing which target is best to track. Thus, if the missile begins tracking the ship at time t, no decoy can break its lock during the interval $[t, t + dt_{m}^{re-target})$. ETA_m is the time at which missile m will reach the ship's immediate vicinity. t_{m}^{appear} is the time at which missile m is first close enough to track the ship. c_d represents the financial cost of deploying decoy d. α, α', and α'' are predefined weighting terms for the objective function. The computational complexity of this formulation is dominated by the integer variables, which yields $O(2^{dmt+dm+dt+dl+d+gmt+m})$.

Equation 4 is a multi-criteria objective function that minimizes a weighted, linear combination of the cost of all decoy deployments, less the total time during which missiles are tracking decoys and the number of missiles successfully guided away from the ship.

Equations 5-12 ensure internal consistency between the variables.Equation 13 ensures that a decoy, if deployed, is active for dt_{d}^{evap} units of time given its timing characteristics. Equation 14 ensures that a decoy is deployed to no more than one location. Equation 15 ensures that, if a decoy is deployed against a missile, its deployment location will be a more attractive target for that missile than the ship. Equation 16 requires that each missile tracks either a ship or decoy while within range. Equations 17-18 force a decoy, if deployed to a location that would cause missile m to impact the ship, to either be deployed after the missile has already been diverted or reached the ship (Equation 17) or to be deployed and evaporate before the missile enters targeting range (Equation 18).

Equation 19 ensures that a missile must be tracking a decoy in the final seconds before it reaches the vicinity of the ship, or else the missile will impact the ship. The duration of this critical period is dependent upon missile dynamics and the target selection process.

Equation 20 ensures that a missile will select the most attractive decoy according to that missile's selection logic. Equation 21 restricts decoy deployments such that the missile heading does not "sweep" across the ship in the final seconds of the missile's flight. If a missile does not have enough time to change its direction toward a newly deployed decoy, that missile will fly into the ship.

Equations 22-25 ensure that the duration of epoch g of missile m while tracking the ship lasts exactly as long as the retargeting time for the missile. Equations 22-23 are akin to Equations 9-11 and relate the start and finish times of ship-tracking epoch g to the decision variable $G_{g,m,t}$. Equation 24 is akin to Equation 12 and relates the start and finish times of ship-tracking epoch g to the decision variable $G_{g,m,t}$. Equation 25 ensures that the tracking time is $dt_m^{re-target}$ if the missile is airborne for at least $dt_m^{re-target}$ seconds. Otherwise, the tracking time is equal to the time before impacting the ship (i.e., $ETA^m - t - 1$). Finally, a term (i.e., $-MG_{g,m,t-1}$) disables the constraint for all t except for the exact moment when t begins tracking the ship.

For the apprenticeship scheduling subroutine's schedulability test, we apply Equations 17-18 as our constraint satisfaction check when testing the feasibility of action τ_i^*, given by applying Equation 3. With regard to tasks within the apprenticeship scheduler's seed solution that are not satisfactorily completed, the MILP can leave those tasks incomplete to start by initially setting $V_m \leftarrow 0$.

Training Dataset for Apprenticeship Scheduling

The algorithm trains the apprenticeship scheduler using a dataset collected from military domain experts playing a serious game that emulates the ASMD problem as formulated in the previous section (Gombolay et al. 2016a). We considered a specific level within the game that requires players to defend against a randomized enemy attack. In this level, 10 missiles are fired at the player's ship from multiple directions, and the player has access to a limitted quantity of five different types of soft-kill weapons to divert these missiles. Although the missile bearings and launch times are fixed, the seeking behavior of the missile is not known a priori.

We collected a dataset of 311 games played by 35 human players across 45 threat configurations, or "scenarios." Of those configurations, only 40 contained a demonstration in which the player completed an entire round. We then subselected the single best demonstration from each of these 40 scenarios. The demonstrators included ASMD professionals with expertise ranging from "generally knowledgeable about the ASMD problem" to "domain experts" with professional experience or ASMD training.

We trained the apprenticeship scheduling algorithm using the following features: The pointwise features for each action included the number of decoys of each type left for possible deployment (i.e., the ammunition). The pairwise features for each action included, for each decoy/missile pair (or null decoy deployment due to inaction), indicators for whether a decoy had been placed such that the missile was successfully distracted by that decoy, whether the missile would be lured into hitting the ship due to decoy placement, or whether the missile would be unaffected by decoy placement. These features are identical to those employed in (Gombolay et al. 2016a).

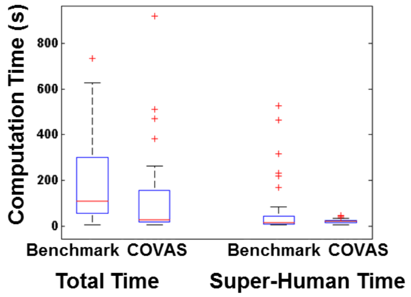

Figure 2: This figure depicts the total computation time for COVAS, as well as the amount of time COVAS required to identify a solution superior to that resulting from a human expert's demonstration.

Results and Discussion

In this section, we empirically validate that COVAS is able to generate optimal solutions more efficiently than state-of-the-art optimization techniques. As a benchmark, we solve a pure MILP formulation (Equations 4-25) using Gurobi, which applies state-of-the-art techniques for heuristic upperbounds, cutting planes and LP relaxation lowerbounds. We set the optimality threshold at 10^{-3}.

Validation Against Expert Benchmark

First, we validate that COVAS can efficiently find optimal solutions, as depicted in Figure 2. To generate each data point, we trained COVAS' apprenticeship scheduling algorithm on demonstrations of experts' solutions to unique ASMD scenarios (save for one "hold-out scenario); we then tested COVAS on this hold-out scenario. We also applied a pure MILP benchmark on this scenario and compared the performance of COVAS to the benchmark. We generated one data point for each unique demonstrated scenario (i.e., leave-one-out cross validation) to validate the benefit of COVAS'.

Figure 2 consists of two performance indicators: The total computation time required for the MILP benchmark and COVAS to solve for the optimal solution is depicted on the left; to the right is the computation time required for the benchmark and COVAS to identify a solution better than that given by a human expert. This figure indicates that COVAS is not only able to improve overall optimization time, but that it also substantially improves computation time for solutions that are superior to those produced by human experts. The average improvement in computation time with COVAS is 6.7x and 3.1x, respectively.

Next, we evaluate COVAS ability to transfer prior learning to more challenging task sets. We trained on a level in the ASMD game in which a total of 10 missiles of vary-

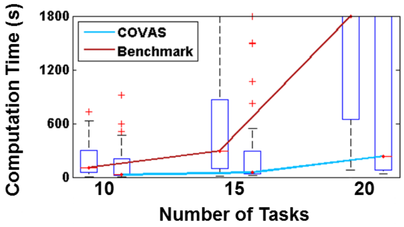

Figure 3: The total computation time needed for COVAS and the MILP benchmark to identify the optimal solution for the tested scenarios.

ing types came from specific bearings at given times. We randomly generated a set of scenarios involving 15 and 20 missiles, with bearings and times randomly sampled with replication from the set of bearings used in the 10-missile scenario.

Figure 3 depicts the computation time required by COVAS and the MILP benchmark to identify the optimal solution for scenarios involving 10, 15 and 20 missiles. We found that the average improvement to computation time with COVAS was 4.6x, 7.9x, and 9.5x, respectively. This evaluation demonstrates that COVAS is able to efficiently leverage the solutions of human domain experts to quickly solve problems twice as large as those the demonstrator provided for training.

Limitations and Future Work

COVAS is able to leverage the power of expert scheduling demonstrations to speed up the computation of provable, globally optimal scheduling solutions. However, the approach is still limited by the quality of the demonstrations provided by the experts and the ability of the apprenticeship scheduling algorithm to generalize the information within those demonstrations. The MILP's computation time is expedited by tight upperbounds (i.e., an initial seed) provided by the apprenticeship scheduling algorithm. If the apprenticeship scheduling algorithm is unable to provide a tight upperbound, the MILP's computation time may not be significantly improved.

In future work, we will explore extensions to the apprenticeship scheduling algorithm to improve its ability to learn from noisy demonstrations. One approach could be to incorporate a trustworthiness metric á la (Zhang 2009) directly into the training of the classifier to uncover a latent action ranking. For example, instead of binary labels, we could reformulate the problem to be one of regression, where positive and negative labels are proportional and inversely proportional, respectively, to the fidelity of the demonstrator.

We also aim to extend COVAS to a stochastic architecture to reason about uncertainty over task assignment characteristics (e.g., missile behavior) and temporal dependencies (e.g., start and finish times, etc.).

Conclusions

In this work, we developed and demonstrated an approach to learning through human demonstrations to efficiently produce optimal solutions for complex real-world scheduling problems. We showed that policies learned from human experts can be used in conjunction with a MILP solver to substantially improve the efficiency of a branch-and-bound search for an optimal schedule. We empirically validated our technique on a dataset collected from human experts solving an anti-ship missile defense problem, which represents the hardest class of scheduling problems. We showed that our approach can substantially improve upon solutions produced by human domain experts, at a rate up to ~ 10 times faster than as an optimization approach that does not incorporate human expert demonstration.

References

Abbeel, P., and Ng, A. Y. 2004. Apprenticeship learning via inverse reinforcement learning. In *Proc. ICML*.

Ahuja, R. K.; Kumar, A.; Jha, K. C.; and Orlin, J. B. 2007. Exact and heuristic algorithms for the weapon-target assignment problem. *Operations Research* 55(6):1136–1146.

Banerjee, A. G.; Ono, M.; Roy, N.; and Williams, B. 2011. Regression-based lp solver for chance-constrained finite horizon optimal control with nonconvex constraints. 131–138.

Berry, P. M.; Gervasio, M.; Peintner, B.; and Yorke-Smith, N. 2011. Ptime: Personalized assistance for calendaring. *ACM Trans. Intell. Syst. Technol.* 2(4):40:1–40:22.

Bertsimas, D., and Weismantel, R. 2005. *Optimization over Integers*. Belmont: Dynamic Ideas.

Cheng, T.-H.; Wei, C.-P.; and Tseng, V. S. 2006. Feature selection for medical data mining: Comparisons of expert judgment and automatic approaches. In *Proc. CBMS*, 165–170.

Chernova, S., and Veloso, M. 2007. Confidence-based policy learning from demonstration using gaussian mixture models. In *Proc. AAMAS*, 233:1–233:8. ACM.

Ciré, A.; Coban, E.; and Hooker, J. N. 2013. Mixed integer programming vs. logic-based benders decomposition for planning and scheduling. In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, 325–331. Springer.

Gombolay, M.; Jensen, R.; Stigile, J.; Son, S.-H.; and Shah, J. 2016a. Decision-making authority, team efficiency and human worker satisfaction in mixed human-robot teams. In *Proc. IJCAI*.

Gombolay, M.; Yang, X. J.; Hayes, B.; Seo, N.; Liu, Z.; Wadhwania, S.; Yu, T.; Shah, N.; Golen, T.; and Shah, J. 2016b. Robotic assistance in coordination of patient care. In *Proceedings RSS*.

Huang, C.-M., and Mutlu, B. 2014. Learning-based modeling of multimodal behaviors for humanlike robots. In *Proc. HRI*, 57–64.

Ijspeert, A. J.; Nakanishi, J.; and Schaal, S. 2002. Movement imitation with nonlinear dynamical systems in humanoid robots. In *Proc. ICRA*, volume 2, 1398–1403.

Konidaris, G.; Osentoski, S.; and Thomas, P. 2011. Value function approximation in reinforcement learning using the fourier basis. In *Proc. AAAI*, 380–385.

Korsah, G. A.; Stentz, A.; and Dias, M. B. 2013. A comprehensive taxonomy for multi-robot task allocation. *IJRR* 32(12):1495–1512.

Odom, P., and Natarajan, S. 2015. Active advice seeking for inverse reinforcement learning. In *Proc. AAAI*, 4186–4187.

Raghavan, H.; Madani, O.; and Jones, R. 2006. Active learning with feedback on features and instances. *Journal of Machine Learning Research* 7:1655–1686.

Ramachandran, D., and Amir, E. 2007. Bayesian inverse reinforcement learning. In *Proc. IJCAI*, 2586–2591.

Ramanujam, V., and Balakrishnan, H. 2011. Estimation of maximum-likelihood discrete-choice models of the runway configuration selection process. In *Proc. ACC*, 2160–2167.

Sammut, C.; Hurst, S.; Kedzier, D.; and Michie, D. 1992. Learning to fly. In *Proc. ICML*, 385–393.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* 529(7587):484–489.

Sutton, R. S.; McAllester, D. A.; Singh, S. P.; Mansour, Y.; et al. 1999. Policy gradient methods for reinforcement learning with function approximation. In *Proc. NIPS*, 1057–1063.

Terrell, A., and Mutlu, B. 2012. A regression-based approach to modeling addressee backchannels. In *Proc. Special Interest Group on Discourse and Dialogue*, 280–289.

Thomaz, A. L., and Breazeal, C. 2006. Reinforcement learning with human teachers: Evidence of feedback and guidance with implications for learning performance. In *Proc. AAAI*, 1000–1005.

Vogel, A.; Ramach, D.; Gupta, R.; and Raux, A. 2012. Improving hybrid vehicle fuel efficiency using inverse reinforcement learning. In *Proc. AAAI*, 384–390.

Wang, Y.-C., and Usher, J. M. 2005. Application of reinforcement learning for agent-based production scheduling. *Eng. Appl. Artif. Intell.* 18(1):73–82.

Wu, J.; Xu, X.; Zhang, P.; and Liu, C. 2011. A novel multi-agent reinforcement learning approach for job scheduling in grid computing. *Future Generation Computer Systems* 27(5):430 – 439.

Zhang, W., and Dietterich, T. G. 1995. A reinforcement learning approach to job-shop scheduling. In *Proc. IJCAI*, 1114–1120.

Zhang, F. 2009. Schedulability analysis for real-time systems with edf scheduling. 58:1250–1258.

Zheng, J.; Liu, S.; and Ni, L. 2015. Robust bayesian inverse reinforcement learning with sparse behavior noise. In *Proc. AAAI*, 2198–2205.

Ziebart, B. D.; Maas, A.; Bagnell, J. A.; and Dey, A. K. 2008. Maximum entropy inverse reinforcement learning. In *Proc. AAAI*, 1433–1438.